VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Android jako bezpečnostní kamera

Android as a Network Security Camera

2013 Lukáš Magnusek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Magnusek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Android jako bezpečnostní kamera
Android as a Network Security Camera

Zásady pro vypracování:

Cílem bakalářské práce je návrh a implementace mobilní aplikace pro platformu Android, fungující jako jednoduchá bezpečnostní kamera. Aplikace bude snímat obrazy zabudovanou kamerou mobilního telefonu a na základě analýzy obrazu vyhodnocovat narušení hlídané zóny.

1. Rešerše aplikací pro střežení pomocí zabudované kamery na platformách Android, iPhone a WP7.
2. Algoritmické možnosti detekce narušení zóny ve snímaném obraze.
3. Implementace vlastní aplikace pro Android s následujícími vlastnostmi: ukládání událostí (snímků s narušitelem) na SD kartu, FTP server; zaslání oznámení o narušení pomocí SMS/MMS nebo E-mailu.
4. Testování aplikace a vyhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] Cay S. Horstmann, Core Java(TM), Volume I--Fundamentals, Prentice Hall, 2007, ISBN-13: 978-0132354769
[2] Reto Meier, Professional Android 2 Application Development, Wrox, 2010, ISBN-13: 978-0470565520
[3] Sayed Hashimi, Pro Android 2, Apress, 2010, ISBN-13: 978-1430226598

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr.Ing. Michal Krumnikl**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA 1

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2013

.....

Rád bych na tomto místě poděkoval panu Mgr. Ing. Michalu Krumniklovi, za jeho rady a návrhy při vytváření této bakalářské práce a také mé rodině za zapůjčení mobilů k testování aplikace.

Abstrakt

Cílem této bakalářské práce bylo navrhnout a v jazyce Java naprogramovat aplikaci pro platformu Android, která by fungovala jako jednoduchá bezpečnostní kamera. Aplikace by měla snímat obrazy z integrované kamery na mobilu či tabletu a následně tyto obrazy analyzovat, zda byl detekován pohyb či nikoli. Při detekci pohybu by aplikace měla být schopna vyfotit aktuální snímek a nahrát video aktuálně střežené zóny. Dále by měla aplikace umět zaslat uživateli SMS, E-mail, nebo by měl být poslán vyfocený snímek na FTP server.

Klíčová slova: Android, bezpečnostní kamera, obraz, pohyb, detekce pohybu

Abstract

The main goal of this bachelor thesis was to design and in Java language program application for the Android platform, that would work as a simple security camera. The application should be able to capture images from the built-in camera on mobile-phone or tablet, and then analyze the images, whether motion has been detected or not. When application detect motion, it should be able to take a picture of the current image and upload currently guarded zone video. In addition, the application should be able to send SMS, E-mail, or should the photo image be sent on the FTP server.

Keywords: Android, security camera, image, movement, motion detection

Seznam použitých zkratek a symbolů

CCD	–	Charge Coupled Device
CMYK	–	Cyan, Magenta, Yellow, blacK
FTP	–	File Transfer Protocol
GNU	–	GNU is Not UNIX
GPL	–	General Public License
GPS	–	Global Positioning System
HSB	–	Hue, Saturation, Brightness
HSV	–	Hue, Saturation, Value
HTTP	–	HyperText Transfer Protocol
IBM	–	International Business Machines
JDK	–	Java Development Kit
MMS	–	Multimedia Messaging Service
NNTP	–	Network News Transfer Protocol
NTP	–	Network Time Protocol
OS	–	Operating System
RGB	–	Red, Green, Blue
RGBA	–	Red, Green, Blue, Alfa
SD	–	Secure Digital
SDK	–	Software Development Kit
SMS	–	Short Message Service
sRGB	–	standard Red, Green, Blue
YUV	–	Luminance (Y), blue–luminance (U), red–luminance (V)
Wi-Fi	–	Wireless Fidelity
WVGA	–	Wide Video Graphics Array

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

S příchodem chytrých telefonů na trh již lidé přestali vnímat mobilní telefon pouze jako věc, ze které se dá volat nebo posílat SMS zprávy. Již v roce 1992 se firma IBM pokusila o tzv. první chytrý telefon s názvem IBM Simon. Byl drahý, proto si jej mohli koupit pouze bohatší lidi. Avšak s porovnáním s dnešními telefony neuměl v podstatě skoro nic. Až o několik let později, s rozvojem nových technologií se začaly tyto přístroje prodávat ve větší míře. V současnosti se chytré telefony prodávají častěji než klasické „hloupé“ telefony. Jako chytrý telefon můžeme označit přístroj, který disponuje technologiemi jako : souborovým manažerem, internetovým prohlížečem, emailovým klientem, multimediálním přehrávačem, kamerou atd. Také využívá vlastního operačního systému, a v důsledku toho se strhl konkurenční boj mezi výrobci těchto operačních systémů. Takovými operačními systémy jsou třeba: Bada, Windows Phone, iOS, Android. Díky svému rychlému růstu a open source licenci jsem si pro bakalářskou práci vybral právě Android.

V dnešní moderní době, kdy se krade téměř vše a zabezpečovací zařízení do domů a bytů jsou velmi drahé, rozhodl jsem se tedy přijít na levnější alternativu střežení objektu a využil jsem integrované kamery v chytrém telefonu.

Úkolem této bakalářské práce bylo naimplementovat aplikaci na střežení zóny a detekci pohybu v ní. Aplikace by měla snímat obraz z integrované kamery na mobilu nebo tabletu a zaznamenávat v určitých časových intervalech snímky. Tyto snímky by následně porovnávala mezi sebou a zjišťovala procentuální podobnost těchto snímků. Následně by se ze snímků určilo, zda byl zachycen pohyb či nikoli. Při detekci pohybu potřebujeme dát uživateli aplikace najevo, že byl zachycen pohyb, a to formou vyfocení fotky, nebo nahráním videa aktuálně střežené zóny. Dále si může uživatel zvolit, zda chce při zachycení pohybu dostávat textové upozornění formou textové zprávy SMS, nebo E-mailem. Také si může zvolit možnost nahrání vyfocené fotky na předem zvolený FTP server. V neposlední řadě bude aplikace disponovat režimem alarmu, kdy se po zachycení pohybu spustí siréna na mobilu.

V následujících kapitolách zhodnotím a srovnám již dostupné aplikace na detekci pohybu v obraze, vysvětlím způsoby prezentace digitálního obrazu a možné metody detekce pohybu v obraze. Následně popíši princip mnou vytvořené aplikace, její implementaci a na závěr testování a dosažené výsledky této aplikace.

2 Dostupné aplikace

V této kapitole je shrnuto několik již dostupných aplikací pro platformy Windows Phone, iOS a Android. Je zde stručný popis ke každé aplikaci, jejich ukázky a na závěr srovnání s mojí aplikací.

2.1 Aplikace pro Windows Phone

Následně je stručně shrnut operační systém Windows Phone a možnosti vývoje aplikací v něm. Poté jsou předvedeny aplikace pro tento systém. Tyto aplikace jsou dostupné na stránkách: `http://www.windowsphone.com/cs-cz/store`.

2.1.1 Operační systém Windows Phone

Windows Phone je jedním ze tří nejrozšířenějších operačních systémů pro chytré telefony od firmy Microsoft. Je to přímý nástupce ne moc úspěšného Windows Mobile. Firma Microsoft tento operační systém vydala na přelomu let 2010/2011. O zásadní průlom se postarala firma Nokia, která se počátkem roku 2011 ze strategických důvodu rozhodla nasazovat tento OS do všech svých mobilů místo svého dosavadního OS Symbian. OS je určen pro chytré telefony se specifickými požadavky, např.: WVGA čtyřdotykový multitouch displej, DirectX9 hardwarová akcelerace, GPS, akcelerometr, digitální kamera a další hardwarové požadavky.

Pro vyvíjení aplikací a her nemusela firma Microsoft vymýšlet nový programovací jazyk, ale použila stávající jako C# a Visual Basic s .NET frameworkem. Pro vývoj těchto aplikací je zapotřebí vývojového prostředí Visual Studio 2010 pro samotnou implementaci programu a Expression Blend pro návrh designu aplikace. [3].

2.1.2 Některé dostupné aplikace

- Motion Cam [1]

  Asi doposud nejlepší aplikace pro detekci pohybu na OS Windows Phone. Funguje na verzích 7.5 i 8. Detekuje nejenom pohyb v obraze, ale i změnu hladiny zvuku na mikrofonu. Poté je schopný automaticky poslat E-mail s vyfocenou fotkou střežené oblasti, nebo může začít nahrávat video, které potom uloží na úložiště SkyDrive.

- Camera Motion Detector [2]

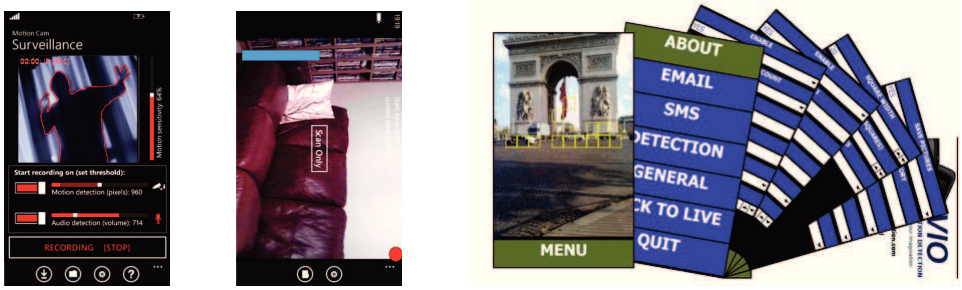
  Poměrně jednoduchá aplikace taktéž pro Windows Phone 7.5 a 8. Při detekci pohybu umožňuje vyfocení fotky nebo pouze uložení události, obsahující datum a čas detekování pohybu. U placené verze, která stojí 21 Kč, lze navíc nastavit čas, kdy bude aplikace střežit zónu.

[1] Ke stažení na: `http://goo.gl/8qPaA`
[2] Ke stažení na: `http://goo.gl/HDIL7`

- VIO Motion Detection [1]

  Aplikace pro starší OS Windows Mobile 5.0 nebo 6.5. Opět jednoduchá aplikace na detekci pohybu, založena na detekci hran v obraze, která umožňuje odeslání E-mailu s obrázkem, nebo poslání SMS až na 2 čísla. Dále umí přehrát výstražný zvuk a uložit obrázek do paměti mobilu.

Na obrázku 1 jsou znázorněny ukázky aplikací pro platformu Windows Phone. Zleva: Motion Cam, Camera Motion Detector, VIO Motion Detection.



Obrázek 1: Ukázky aplikací pro platformu Windows Phone

2.2 Aplikace pro iOS

V této podkapitole je shrnut operační systém iOS a vývoj aplikací v něm. Aplikace jsou dostupné za použití služby iTunes.

2.2.1 Operační systém iOS

iOS je v současnosti nejrozšířenější operační systém a asi největší rival Androidu. Tento název je používán až od čtvrté verze systému, do té doby byl pojmenován jako iPhone OS. Je to operační systém vytvořen společností Apple Inc., původně určen pouze pro mobilní telefony iPhone. Později také pro další zařízení firmy, jako: iPhone, iPod, iPad a dokonce i pro Apple TV. Je to vlastně odlehčená verze systému Mac OS X, který je používán v počítačích firmy Apple, tudíž se jedná o systém UNIXového typu.

Pro vývoj aplikací a her pro tento OS je potřeba mít počítač od společnosti Apple (Macintosh) s operačním systémem Mac OS X ve verzi 10.7, nebo novější. Dále musí být uživatel registrován jako iOS vývojář. Programuje se zde v jazyce C nebo objective-C v prostředí X-Code. Tento program je určen pouze pro OS Mac OS X, avšak jsou už i další programy pro vývoj, jako např. aplikace Flash od společnosti Adobe, pod systémy Windows a Linux. [4].

[1] Ke stažení na: `http://goo.gl/AnEdX`

2.2.2 Některé dostupné aplikace

- Security Cam [1]

  Profesionální bezpečnostní kamera pro iPhone, iPod Touch a iPad, jehož hlavní předností je upload snímků na úložiště Dropbox a Youtube. Kromě detekce pohybu pomocí kamery disponuje aplikace i detekcí hladiny zvuku, kdy při překročení určitého práhu program vyfotí foto, nahraje video, nebo v určitých intervalech začne pořizovat snímky, z nichž je pak možné vytvořit Time-lapse video. V aplikaci lze i nastavit časové razítko u fotek. Program si lze koupit za cenu $9.99 v App Store.

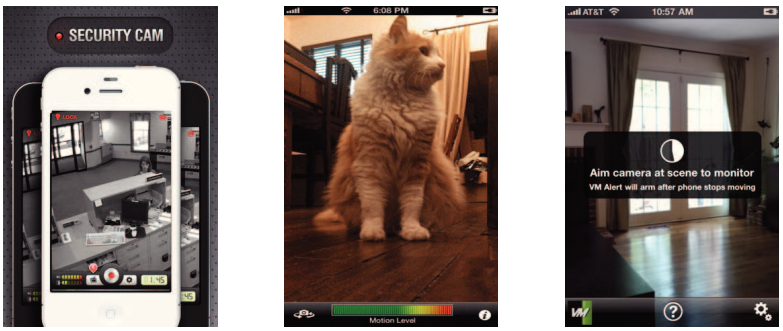
- Motion Detector [2]

  Tuto aplikaci vytvořila firma Senstic jak pro iOS, tak i pro Mac OS X a Windows 8. Jednoduchá aplikace, která při detekci pohybu, pomocí integrované kamery začne nahrávat video, či pošle E-mailové upozornění. V Aplikaci si lze nastavit citlivost detekce střežené zóny nebo délku nahrávaného videa. Aplikace je určena pro iPhone i pro iPad a lze si ji pořídit za $2.99 v App Store.

- VM Alert [3]

  Aplikace pro detekci pohybu z přední nebo zadní kamery iPhonu nebo iPadu. Aplikace je zatím nejpodobnější mnou vytvořené aplikaci. Pro střežení se musí zařízení postavit na místo, kde nedochází k žádným otřesům a po 3-5 sekundách začne aplikace střežit. Tak jak je tomu i v mé aplikaci, zařízení nesmí být v pohybu, jinak nebude zaznamenán pohyb. Lze také nastavit citlivost, kdy je pohyb zaznamenán a kdy ne. Umí ukládat fotky přímo do paměti zařízení a také vydávat zvukový signál při detekci pohybu. Program si lze pořídit za cenu $1.99 v App Store.

Na obrázku 2 jsou znázorněny ukázky aplikací pro platformu iOS. Zleva: Security Cam, Motion Detector, VM Alert.



Obrázek 2: Ukázky aplikací pro platformu iOS

[1] Ke stažení na: `http://goo.gl/KKsi8`
[2] Ke stažení na: `http://goo.gl/wfmUz`
[3] Ke stažení na: `http://goo.gl/F6wvY`

2.3 Aplikace pro Android

Nakonec je předveden OS Android a možnost vývoje aplikací pro tuto platformu. Aplikace jsou ke stažení na: `https://play.google.com/store`.

2.3.1 Operační systém Android

Je to OS vyvíjen firmou Google, která jej odkoupila v roce 2005 a v roce 2008 byla uvedena první verze systému ve verzi 1.0. Je to open source platforma, tzn. software s otevřeným kódem. To pro uživatele znamená, že má přístup ke zdrojovým kódům, které si může dle své libosti upravovat. Celý OS je postaven na otevřeném Linuxovém jádře ve verzi 2.6.

Stejně jako vývoj aplikací pro Windows Phone a iOS, je založen na Host-Target vývojovém prostředí, tzn., že v prostředí, ve kterém se aplikace vyvíjí, jsou odlišné od prostředí, kde aplikace poběží. Samotný kód se píše v Javě. Je proto mít nutností nainstalován JDK, kde jsou knihovny a nástroje právě pro vývoj v tomto jazyce. Dále je třeba mít nainstalováno SDK, což jsou nástroje a knihovny určené přímo pro vývoj aplikací pro Android. Nejčastějším programem pro vývoj aplikací a her pod OS Windows je program Eclipse, avšak je možno i použití i jiných programů i pod OS Linux a Mac OS X. [5].

2.3.2 Některé dostupné aplikace

- Motion detector Pro [1]

  Aplikace od firmy MVA. Program má velmi pěkně zpracované grafické prostředí a hlavním účelem tohoto programu je detekce pohybu pomocí integrované kamery. V aplikaci lze nastavit práh, kdy bude detekován pohyb a disponuje možnostmi jako : odesílání SMS, E-mailu a funkcí alarmu, kdy je možno nastavit odemknutí telefonu pomocí pinu. Umožňuje také ukládání snímků na cloudové úložiště.

- Mobile Alarm System [2]

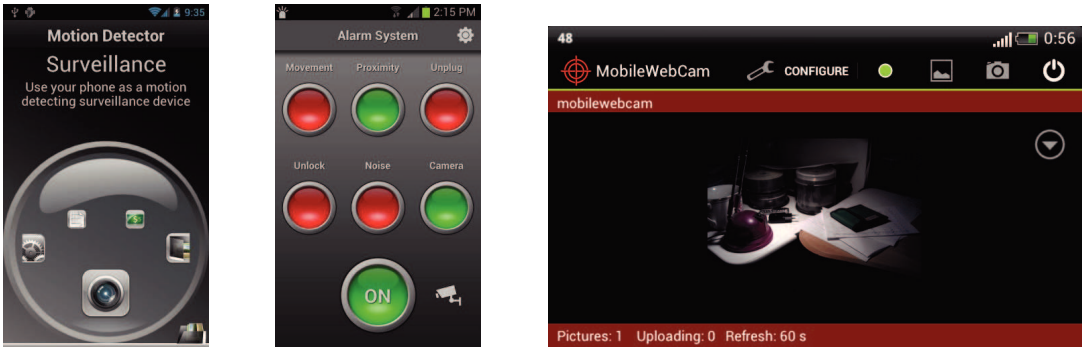
  Aplikace od firmy Focusware. Hlavním účelem aplikace je detekování aktivit z různých senzorů. Mezi tyto aktivity patří : pohyb telefonu, rekce na proximity senzor (senzor přiblížení), odpojení telefonu od nabíječky, odemknutí telefonu, detekci hlasitého zvuku a hlavně detekci pohybu pomocí integrované kamery. Je zde také možnost odemknutí telefonu pomocí předem nastaveného pinu.

- Mobile WebCam [3]

  Aplikace od vývojáře Michaela Haara. Určena jako webová kamera, ale funguje i jako bezpečnostní kamera. Snímky je možno pořizovat v pravidelném intervalu, nebo až při detekci pohybu. Aplikace umožňuje ukládat snímky na SD kartu, FTP server, Dropbox a posílat na E-mail. Také disponuje velmi snadným použitím služby SENSR.NET, což je webové rozhraní na ukládání zachycených snímků.

[1] Ke stažení na: `http://goo.gl/cT5qU`
[2] Ke stažení na: `http://goo.gl/kUV3j`
[3] Ke stažení na: `http://goo.gl/x9diF`

Na obrázku 3 jsou znázorněny ukázky aplikací pro platformu Android. Zleva: Motion detector Pro, Mobile Alarm System Pro, Mobile WebCam.



Obrázek 3: Ukázky aplikací pro platformu Android

2.4 Srovnání aplikací

Nakonec je v tabulce 1 zobrazeno srovnání aplikací s mým vytvořeným programem. Jsou zde srovnány pouze aplikace určené pro platformu Android, jelikož jsem neměl možnost reálného testování na platformách Windows Phone a iOS. Tudíž jsem nemohl zcela přesně zjistit všechny funkce dané aplikace.

	Motion detector	Mobile Alarm	Mobile WebCam	Má aplikace
Zdarma	Ano	Ne	Ano	Ano
Uložení fotky	Ano	Ano	Ano	Ano
Nahrání videa	Ne	Ne	Ano	Ano
Odesílání SMS	Ano	Ano	Ne	Ano
Odesílání MMS	Ne	Ne	Ne	Ne
Odesílání mailu	Ano	Ano	Ano	Ano
Upload FTP	Ne	Ne	Ano	Ano
Upload HTTP	Ne	Ne	Ano	Ne
Funkce alarmu	Ano	Ano	Ne	Ano

Tabulka 1: Srovnání aplikací pro Android

Z tabulky můžeme vyčíst jednotlivé funkce aplikací, tudíž i jejich použitelnost. Z tohoto pohledu si má aplikace vede nejlépe.

Všechny URL adresy pro stažení aplikací musely být kvůli jejich délce zkráceny službou Google url shortener. Služba je dostupná na: `http://goo.gl/`. Všechny tyto odkazy jsou vytvořeny jako stálé.

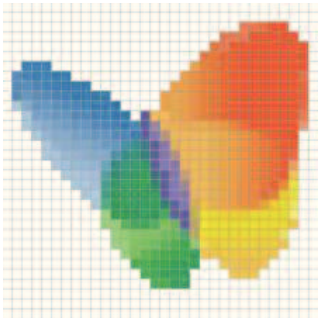
3 Detekce pohybu v obraze

V této kapitole bude nejprve popsána reprezentace digitálního obrazu, barevné modely pro zaznamenání tohoto obrazu a následně popsány metody pro korekci obrazu a metody pro detekci pohybu v obraze.

3.1 Digitální obraz a jeho reprezentace

Obraz, tak jak jej vidíme lidským okem, není nic jiného než světlo v určitém barevném spektru. Lidské oko dokáže zachytit světlo o vlnové délce okolo 350 - 750 nm. My ovšem tento námi viditelný obraz potřebujeme převést do digitální podoby.

Digitálním obrazem rozumíme reprezentaci obrazové informace reálného světa, jedná se tedy o převod z analogové formy obrazu do podoby digitální. Tato forma obrazu je uložena v digitální paměti ve formě jedniček a nul. Jelikož mluvíme o tzv. rastrové grafice, musí být obraz rozdělen do několika bodů. Těmto bodům říkáme pixely (složením slov picture a element). Jsou to nejmenší, nedělitelné a bezrozměrné jednotky v digitálním obraze. Pokud mluvíme o černobílém obraze, je v jednom pixelu uložena právě jedna informace a to o jasu tohoto pixelu. Pokud mluvíme o barevném obraze, je tento pixel znázorněn jedním z barevných modelů, o kterých si povíme později. Tento digitální obraz, zobrazen na obrázku 4 lze následně zapsat jako mapu pixelů, nebo lze také vyjádřit jako funkci $f(x, y)$, kde x, y jsou prostorové souřadnice. [6].



Obrázek 4: Digitální obraz výjádřen pixely

3.1.1 RAW formát

RAW formát jsou syrové, nezpracované data digitálního obrazu, většinou velké kapacity. Jsou to digitální informace, zachycené přímo snímačem na kameře nebo videorekordéru. Vyjímkou nejsou ani integrované kamery v mobilních zařízeních. Tento formát si můžeme představit podobně jako negativ u fotek. Soubor obsahuje 2 typy informace: obrazové pixely a metadata celého obrázku. Z těchto dat se následně raw konvertorem převede soubor na RGB složky a poté na jakýkoliv obrazový formát jako .jpg, .bmp, .png atd.

3.2 Barevné modely

Abychom mohli vyjádřit pixel barvou, potřebujeme si definovat tzv. barevné modely. Většinou se jedná o mísení základních barev do barvy výsledné. V praxi se používají dvě metody na mísení těchto složek a to: [6]

- Aditivní míchání barev

  Způsob, kdy smícháním tří barev (zelené, červené, modré) vznikne součtem těchto barev, barva výsledná. Tohoto míchání využívá RGB model a používá se u monitorů.

- Subtraktivní míchání barev

  Využívá se také tří barev (žluté, azurové, purpurové). Oproti aditivnímu míchání se složky nesčítají, ale odečítají, tzn., že světlo prochází těmito složkami barev, ale je stále více pohlcováno. Tento způsob využívá CMYK model a je využit u tiskáren.

3.2.1 RGB

Jedná se o aditivní míchání barev. Tento model je nejstarším barevným modelem v informatice, protože se ho již využívalo u prvních barevných televizorů. Například složení barvy červené a zelené nám vznikne barva žlutá. Nastavením jasu červené nebo zelené složky nám vznikne výsledný odstín žluté barvy. Ale jelikož model RGB nedefinoval úplně přesně reálné barvy, musely se zavést standarty jako sRGB, Apple RGB, ProPhoto RGB a Adobe RGB.

3.2.2 CMYK

Jedná se o subtraktivní míchání barev. Tyto barvy jsou vlastně úplným opakem barev červené, zelené a modré. Tento model vznikl zejména kvůli problémům, kde se nedal model RGB využít, např. u tiskáren. K tomuto modelu musela být následně přidána černá barva (K), jelikož kvůli smíchání všech tří barev by se plýtvalo inkoustem. Největší nevýhoda tohoto modelu je, že obraz podaný v RGB není naprosto shodný s obrazem podaným v CMYK a to právě kvůli rozdílně dosažitelným barvám v těchto dvou modelech. Můžeme tedy říci, že obraz na monitoru nebude vypadat stejně jako na papíře.

3.2.3 HSV

Někdy také model HSB (hue, saturation, brightness). Tento model vznikl, jelikož odpovídá nejvíce představě obrazu reálného světa, tedy tomu jak člověk tuto barvu vnímá. Skládá se ze tří složek, ovšem zde už se barvy nemíchají, jak tomu bylo u předchozích modelů. Představme si tento model jako válec, kdy po obvodu tohoto válce je definován odstín barvy, od středu po okraj válce je definována sytost barvy a ve výšce válce je definován jas, neboli můžeme říct, že s klesající výškou přidáváme do výsledné barvy, barvu černou.

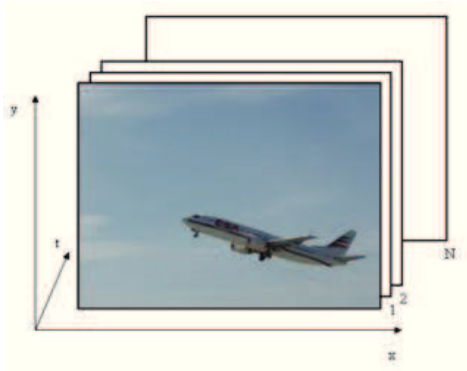
Existují samozřejmě i další barevné modely, ale v této práci nemá smysl se o nich dále zmiňovat.

3.3 Pohyb

Pod pojmem pohyb, jiným slovem také lokomace, rozumíme stav hmotných objektů, které mění svou polohu, tvar, popř. velikost. Rozeznáváme několik druhu pohybů, jako tepelný, elektrický, biologický atd. Nás ale bude zajímat pohyb mechanický.

3.3.1 Analýza pohybu

V současné době má detekce pohybu velké využití v mnoha spektrech života. Používá se například při střežení bank, průmyslových i soukromých objektů. Hlavním principem pro analýzu a detekci pohybu je srovnávání dvou snímků. U jednoho snímků nelze detekovat žádný pohyb, lze pouze zjistit polohu samotných hmotných objektů. Tomuto snímku se říká statický obraz, který si můžeme opět znázornit funkcí $f(x, y)$. Abychom mohli analyzovat pohyb, potřebujeme obraz dynamický. Dynamický obraz je již posloupnost obrazu statických, mezi kterými známe časy jejich pořízení. Z tohoto obrazu můžeme následně kromě polohy vyčíst i rychlost, zrychlení a směr. Tento obraz je znázorněn na obrázku 5 a můžeme si ho znázornit funkcí $f(x, y, t)$, kdy t je čas pořízení snímku. [9, 10].



Obrázek 5: Dynamický obraz

3.3.2 Vztahy mezi objekty

Existují 2 základní vztahy mezi kamerou a objektem ve střežené zóně :

- Kamera v klidu, objekt v klidu
- Kamera v pohybu, objekt v klidu
- Kamera v klidu, objekt v pohybu
- Kamera v pohybu, objekt v pohybu

Pro úspěšnou detekci pohybu daného objektu musíme tedy vědět, jestli snímací zařízení bude v pohybu, či nikoli a podle toho následně zvolit metodu na detekci pohybu.

3.4 Metody pro zpracování dynamického obrazu

Standartní postup při detekci pohybu je rozdělení obrazu na jeho pozadí a popředí, kde pozadí reprezentuje část, na které chceme zjišťovat změny a právě popředí tyto změny v sobě nese. Bohužel ne vždy je pozadí úplně stejné vůči předcházejícímu obrazu. To je způsobem například rychlými změnami v počasí, větrem nebo změnou světla např. při rozednívání nebo stmívání. Ovšem hlavním problémem je šum v obraze. Ten je dán většinou ne úplně dokonalým snímačem CCD v kameře nebo mírným pozměněním několika pixelů v obraze. Tento šum chceme v co největší míře odstranit a to některou z těchto filtrací: [7, 8]

- Metoda průměrování

  Nejjednodušší metoda pro odstranění šumu v obraze. Princip spočívá v tom, že pixel, který chceme v obraze nahradit, se nahradí aritmetickým průměrem všech osmi okolních pixelů. Bohužel obraz může být v některých případech po filtraci rozmazán, tudíž se ho moc nedoporučuje použít při detekci hran.

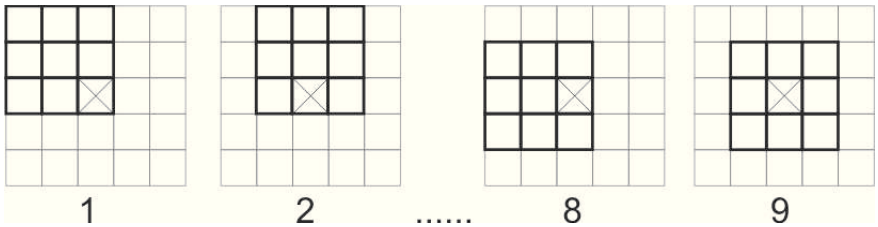
- Gaussův filtr

  Vysoce efektivní metoda pro odstranění šumu, ale taktéž vede k rozmazání obrázku. Pracuje na principu gaussovy funkce. Tato funkce je znázorněna ve vzorci 1.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-} \frac{x^2 + y^2}{2\sigma^2} \tag{1}$$

- Metoda rotující masky

  Metodu rotující masky spočívá v tom, že se filtrovanému bodu pokusí najít takové okolí, ke kterému pravděpodobně patří. Na obrázku 6 je předveden princip rotující masky. Okolí bodu má zde rozměr 5x5 bodů a samotná maska 3x3 body. Maska rotuje kolem filtrovaného bodu a vybere to okolí, ve kterém je rozptyl jasu nejmenší. Tato metoda má výhodu, že nerozmazává výsledný obraz a je tudíž vhodný pro detekci hran.



Obrázek 6: Princip rotující masky

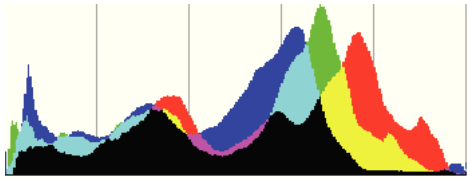
- Metoda mediánu

  Efektivní metoda pro odstranění šumu. Tato metoda je podobná průměrování, s tím rozdílem, že nahrazovaný bod není spočten aritmetickým průměrem všech okolních bodů, ale jejich mediánem. To je vlastně hodnota jasu, ležící uprostřed ve vzestupné posloupnosti jasů těchto okolních bodů.

Toto jsou některé z často používaných filtrací obrazů, po této filtraci už můžeme přejít k samotné detekci pohybu jednou z několika následujících metod.

3.4.1 Porovnávání histogramů

Je to nejjednodušší metoda pro detekci pohybu. Jelikož je výpočetně málo náročná, tak patří mezi nejvíce používané metody v praxi. Funguje na principu porovnávání histogramů aktuálního a referenčního snímku. Histogram je takové znázornění obrazu, kde každému bodu náleží jedna jasová hodnota (0 - 255), tato hodnota je pak uložena do jednorozměrného pole spolu s počtem výskytů tohoto jasu v obraze. V případě barevného obrazu musíme mít 3 histogramy, pro každou složku z barevného modelu RGB nebo jeden společný RGB histogram. Grafické znázornění histogramu je na obrázku 7. [9, 10].



Obrázek 7: RGB Histogram

Tato metoda má ale bohužel několik nevýhod. Jednou z nevýhod je šum. Tento jev se projeví tak, že na aktuálním a referenčním snímku je šum jiný, tudíž pro každý snímek jiný histogram a v obecné implementaci by byl detekován pohyb. Ale díky malé výpočetní náročnosti na vygenerování histogramů, můžeme předem aplikovat jeden nebo několik filtrů a tímto bychom problému předešli.

Dalším problémem je nemožnost zjistit, kde byl pohyb zaznamenán. A to díky porovnávání histogramů celého snímku. Tomuto se dá předejít, pokud obraz rozdělíme do několika podobrazů. Pak budeme moct zjistit, v jakém podobraze byl detekován pohyb.

Ovšem hlavním problémem této metody je, že většinou nezachytí pohyb objektů, které jsou jak na aktuálním, tak referenčním snímku. Můžeme si to představit tak, jako kdybychom snímali zónu terária s křečkem uvnitř. Tento křeček by po teráriu běhal, ale tato metoda by nezachytila žádný pohyb, jelikož křeček ve snímané zóně představuje stejný počet barev v histogramu. Pohyb by byl zachycen, kdybychom snímali terárium bez křečka a následně ho do něj přidali. Toto se dá částečně také řešit segmentací obrazu.

Tato metoda je vhodná do míst se stálým osvětlením, kde budeme detekovat pohyb v zóně, do které čekáme, že někdo přijde nebo přijede.

3.4.2 Rozdílová metoda

Velice jednoduchá metoda na detekci pohybu, která pouze porovnává každý bod jednoho obrazu s korespondujícím bodem obrazu druhého. Mějme statický obraz $f1(x, y)$ a statický obraz $f2(x, y)$. Při použití rozdílové metody nám vznikne výsledný rozdílový obraz $d(x, y)$, kde bod je znázorněn rozdílem bodu prvního a druhého obrazu. Pokud není rozdíl žádný, má výsledný bod hodnotu 0 a v obraze ho můžeme znázornit černou barvou. Na obrázku 8 je znázorněn referenční snímek zóny, aktuální snímek zóny s autem a výsledný rozdílový snímek po aplikování rozdílové metody. [9, 10].

Obrázek 8: Rozdílová metoda

V tomto případě by byl pohyb zachycen, jelikož aspoň z části jsou oba obrazy (referenční a aktuální) rozdílné.

Jelikož srovnáváme každý bod v obraze s korespondujícím bodem v obraze druhém, je tato metoda v obecné implementaci dosti výpočetně náročná. Pokud porovnáváme černobílý obraz, porovnávají se pouze odstíny těchto bodů. Ovšem za předpokladu, že fotka není příliš velká. Zde o velké výpočetní náročnosti moc hovořit nemůžeme. Problém nastává až u barevných obrazů, jelikož musíme srovnávat více složek barvy. Proto existuje několik způsobů jak tomuto předejít, a to:

- Převod z RGB modelu na HSV model

  Tento způsob má tu výhodu, že obraz neztratí své barevné informace a pro funkci rozdílové metody by stačilo porovnávat pouze 1 nebo 2 složky tohoto modelu.

- Převod obrazu do menší bitové hloubky

  Tento způsob má nevýhodu, jelikož ztratíme určité barevné informace a obraz se tak stává méně kvalitnějším. Ovšem rychleji se pak budou porovnávat body mezi snímky.

Bohužel i tato metoda s sebou nese nějaká úskalí. Opět zde nemůžeme zjistit zrychlení a směr objektu. Za to už můžeme zjistit, kde přesně k pohybu v obraze došlo a to např. vhodnou implementací, kdy bychom zjišťovali, kde je u sebe více rozdílných bodů. Tímto bychom dostali polohu objektu v obraze, kde k pohybu došlo.

Ovšem největší výhodou této metody je, že většinou není potřeba obraz filtrovat před zpracováním, jelikož určitá filtrace probíhá už přímo při detekci pohybu. Tato filtrace spočívá v tom, že se při porovnávání snímků určí práh, který určí, zda bude bod obrazu ještě stejný, či nikoli.

3.4.3 Detekce hran

Není to vlastně úplně metoda pro zjištění pohybu v obraze, ale spíše filtrace obrazu. Ta spočívá v tom, že se v obraze snaží najít hrany. Hranou rozumíme okraje hmotných objektů. Tyto hrany pak mohou být buď zašumělé, které vznikly velkým šumem v daném místě, nebo hrany teoretické, které jsou již většinou reálnými okraji objektů. Hrana je zachycena, pokud se v obraze hledají místa, kde je patrná velká změna jasu. Pokud jsou hrany zachyceny, poskytne nám to určité rozdělení obrazu na objekty, čímž je ulehčena detekce pohybu těchto objektů. [9, 10].

Jako výhoda této metody je značné odstranění šumu. Pokud tento šum není detekován jako zašumělá hrana, pak zmizí šum z obrazu úplně. Po aplikaci detekce hran na obraz je vhodné použít jedné z předcházejících metod, abychom byli schopni detekovat pohyb. Další výhoda této metody spočívá v tom, že při několika-násobné filtraci, nám z obrazu zbude pouze bílé pozadí a černé hrany. Po aplikaci vhodné metody, již můžeme zjistit zrychlení a směr těchto objektů.

Na obrázku 9 je znázorněn původní obraz, obraz s hranami a samotné hrany.

Obrázek 9: Detekce hran

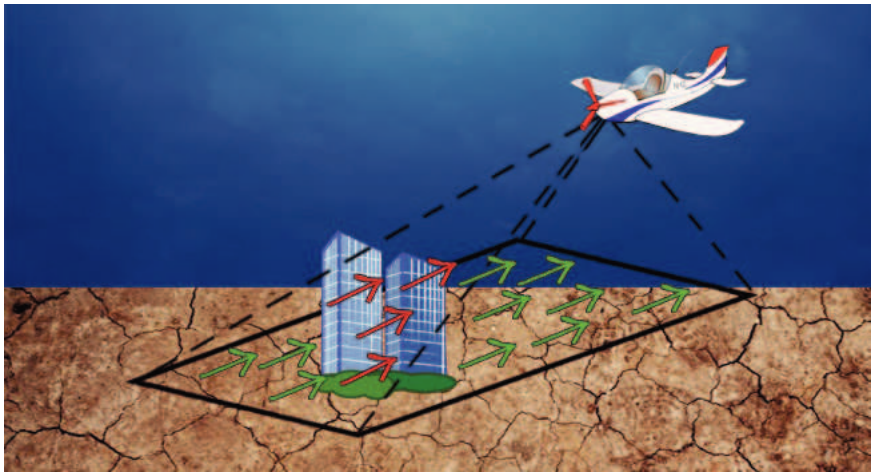
Jelikož hranu definujeme jako velkou změnu jasu v okolí bodu, tak je zjevné, že v tomto místě bude velká hodnota derivace jasové funkce a maximální hodnota derivace bude ve směru kolmo na tuto hranu. Tyto hrany se poté hledají tak, že tuto derivaci aproximují pomocí operátoru, zpracovávajícího 2 funkce, kterému říkáme konvoluce. Tato konvoluce má jádro, což můžeme chápat jako tabulku, nebo matici typu $m * m$. Tato matice se následně položí na každé místo obrazu a každý pixel se vynásobí číslem v příslušném řádku, sloupci. Nakonec se provede součet všech těchto čísel a vznikne nám jeden nový bod.

Nejjednodušší konvoluční jádra jsou $(-1, 1)$ a $(-1, 1)^T$, nebo $(-1, 0, 1)$ a $(-1, 0, 1)^T$. A složitější jádra pojmenovaná jako : Robertsův operátor, Sobelův operátor, Kirschův operátor, Robinsonův operátor atd.

3.4.4 Metoda optického toku

Tato metoda zjišťuje pohyb pixelů v obraze. Každý tento pixel v obraze je definován jako dvojrozměrný vektor, který nám určuje směr a rychlost pohybu. Tudíž nespornou výhodou této metody je, že můžeme z obrazu vyčíst polohy objektů, jejich směr, rychlost a nakonec zrychlení. To se například hodí k předpovědi, kde budou objekty teoreticky směřovat, nebo naopak kde se v minulosti nacházely. Avšak tato metoda patří mezi výpočetně nejnáročnější, tudíž její použití pro domácí, či veřejné účely se moc nehodí. A jako jedna z několika mála metod nám detekuje pohyb i pokud je detektor, popř. kamera v pohybu. Díky tomu tato metoda může posloužit armádě, či letectvu. [9, 10].

Vezměme si příklad, kdy letí letadlo nad zemí a detektor má zabudovaný na podvozku letadla. Tento detektor snímá zem a následně na ní určuje optický tok. Kdyby letadlo letělo v ideální rovině, bude mít tento optický tok stálý směr a stálou rychlost. Ovšem pokud přeletí nad budovou, či kopcem, tak rychlost vektoru zvýší, jelikož momentálně snímaná oblast (budova, kopec) je blíže detektoru než zem.



Obrázek 10: Ukázka použití metody optického toku

Ne obrázku 10 lze vidět letadlo letící nad krajinou se zabudovanou kamerou. Letadlo snímá zem, a jelikož je budova vyšší než země, budou zde obrazové body znázorněny vektorem s větší rychlostí, ale stejným směrem oproti obrazovým bodům na zemi.

Jak jsme se již předem zmínili, tak dynamický obraz je definován jako jasová funkce $f(x, y, t)$. Jelikož tato metoda zachycuje všechny body v obraze v čase dt, tak si můžeme tyto hodnoty vložit do Taylorovy řady:

$$f(x + dx, y + dy, t + dt) = f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt \tag{2}$$

$$f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt = f(x, y, t) + f_x dx + f_y dy + f_t dt \tag{3}$$

Za předpokladu stálého osvětlení obrazu při pohybu objektu následně platí:

$$f(x + dx, y + dy) = f(x, y, t) \tag{4}$$

$$-f_t = f_x \frac{dx}{dt} + f_y \frac{dy}{dt} \tag{5}$$

A cílem je také vypočítat rychlost tohoto vektoru následovně:

$$v = (dx/dt, dy/dt)^T \tag{6}$$

3.4.5 Detekce významných bodů

Je to metoda, která využívá z části metody podobnou detekci hran a rozdílové metody. Patří tudíž mezi ty výpočetně náročnější. Skládá se ze dvou částí, a to: [9, 10].

- Nalezení významných bodů

  Musí se nalézt významné body v obraze, podobně jako u metody detekce hran. Vyberou se ty body, které mají ve svém okolí výrazně jiný jas, popř. jsou body hraničními. Tyto body mohou být vrcholy, nebo okraji objektu. Pro nalezení těchto bodů se využívá Moravcův operátor:

  $$f(i, j) = \frac{1}{8} \sum_{k=i-1}^{k=i+1} \sum_{e=j-1}^{j+1} |g(k, e) - g(i, j)| \tag{7}$$

  Zajímají nás ovšem pouze významné body, které mění svůj pohyb vůči dvěma obrazům. Pro nalezení takových bodů je vhodné použít rozdílové metody.

- Nalezení příslušného bodu na druhém obraze

  Pokud jsme našli tyto body, potřebujeme si určit tzv. rychlostní pole. To je pole, kde se vyskytují vhodní kandidáti na příslušné body z prvního obrazu. Toto pole se snažíme co nejvíce minimalizovat a to že jsou např. vyřazeny ty body, které porušují určitá pravidla v obraze. Například, že se nemůže bod z prvního obrazu objevit na druhém obraze o 10 metrů dále, při pořízení druhého snímku 0,5 sekundy od prvního atd. Dále je třeba si ohodnotit tyto body pravděpodobnosti, že jsou to právě tyto korespondující body z obrazu prvního. Tyto body se ohodnotí např. na základě společného pohybu bodů. To znamená, že pokud je nalezeno více významných bodů na jednom místě, tak největší ohodnocení pravděpodobnosti dostanou právě ty body, které budou nalezeny také na společném místě. Algoritmus končí, když jsou nalezeny všechny korespondující body z obrazu prvního na obraze druhém.

Závěr

Byl uveden princip zaznamenání digitálního obrazu a poté byly stručně vysvětleny některé metody pro detekci pohybu. Nakonec jsem si pro vytvářenou aplikaci vybral metodu podobnou metodě rozdílové, díky její jednoduchosti.

4 Popis aplikace

V této kapitole se budu věnovat podrobnému popisu mnou vytvořené aplikace. Popíšu její hlavní princip, funkce aplikace a možnosti nastavení.

4.1 Obecný popis

Aplikace funguje jako jednoduchá bezpečnostní kamera, která snímá obraz pomocí integrované kamery v mobilu. Tento obraz snímá v určitém intervalu, který je v nastavení aplikace možno změnit. Následně algoritmus aplikace porovnává snímky (vždy aktuální a přechozí snímek) a určuje procentuální shodu zachycených snímků. Tuto shodu lze také nastavit v nastavení. A pokud je shoda již pod nastaveným prahem, aplikace vyvolá určitou událost, která nám již značí detekci pohybu. V základním režimu, při detekci pohybu máme několik možností, jak tuto skutečnost oznámit:

- Vyfocení fotky (Uložení na SD kartu zařízení)
- Nahrání videa (Uložení na SD kartu zařízení)
- Zaslání textové zprávy
- Zaslání E-mailu i s vyfoceným snímkem
- Upload vyfoceného snímku na FTP server

Jako alternativu zde máme možnost spustit střežení jako Alarm. Aplikace také střeží zónu, avšak při detekci pohybu už nenastane některá z předcházejících událostí, ale pouze se spustí siréna.

4.2 Požadavky

Pro spuštění aplikace je zapotřebí zařízení s operačním systémem Android, minimálně ve verzi 2.3 Gingerbread. To zajišťuje správný a bezproblémový chod této aplikace. Dále musí zařízení mít integrovanou kameru a také akcelerometr, jelikož je využit při detekci, zda je zařízení v pohybu, či nikoli.

4.2.1 Oprávnění aplikace

Abychom mohli aplikaci nainstalovat a používat, potřebujeme při její instalaci povolit následující oprávnění :

- `CAMERA`

  Oprávnění k použití integrované kamery.

- `RECORD_AUDIO`

  Oprávnění na nahrávání zvuků z mikrofonu zařízení. Je potřeba při nahrávání videa.

- `INTERNET`

  Umožňuje aplikaci otevírat síťové sockety. Používá se při uploadu na FTP server a při posílání E-mailu.

- `ACCESS_NETWORK_STATE`

  Zjištění stavu sítě. Při posílání E-mailu nebo uploadu na FTP server si prvně zjistíme, zda je k dispozici síť Wi-Fi.

- `WRITE_EXTERNAL_STORAGE`

  Umožňuje zápis na externí uložiště. Použití při ukládání fotek a videa.

- `SEND_SMS`

  Oprávnění, jenž nám umožňuje posílat SMS zprávy, bez zásahu uživatele.

- `WAKE_LOCK`

  Oprávnění, které zabraňuje telefonu, aby přešel do režimu spánku. Využívá se zde, aby aplikace neusnula a stále střežila zónu.

4.3 Popis funkcí a nastavení

V této části se budu věnovat popisu funkcí. Bude předvedeno, jak aplikace vypadá a co vše je možné nastavit.

4.3.1 Hlavní menu

Zobrazí se při spuštění aplikace. Hlavní menu je koncipováno tak, aby se v něm orientoval i nezkušený uživatel. Po levé straně je logo aplikace s názvem Security Camera. Dole je vypsána aktuální verze programu a vedle autor aplikace. Po straně pravé jsou již tlačítka pro určitou funkci programu. Na obrázku 11 je zobrazeno grafické zpracování aplikace jak na výšku, tak na šířku zařízení.

Obrázek 11: Hlavní menu aplikace

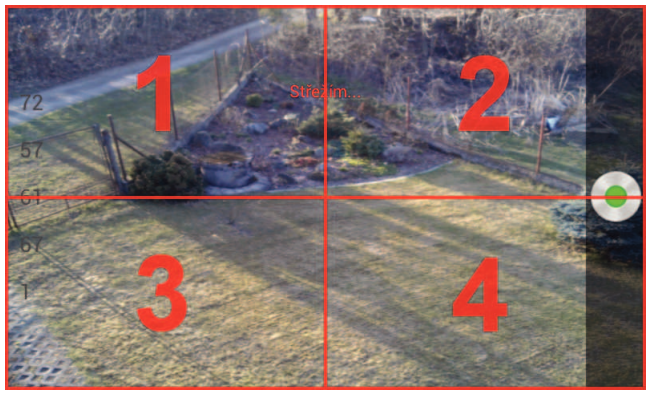
4.3.2 Střežení zóny

Střežení zóny se spouští prvním tlačítkem Spustit. Následně se spustí aktivita pro samotnou detekci pohybu, kde se při zachycení pohybu spustí některé z předem zmiňovaných událostí.

Pozadí aktivity je reprezentováno přímým výstupem z kamery zařízení, tudíž vidíme obraz, který budeme střežit. Na pravé straně je červené tlačítko pro spuštění střežení, které se po stisknutí změní na zelené, což indikuje probíhající střežení. Z aplikace se následně nedostaneme zpět. Nejprve se musí střežení ukončit opětovným stisknutím spouště.

Pro správnou detekci pohybu je zapotřebí před samotným zapnutím střežení postavit zařízení (telefon) na plochu, kde nedochází k žádným otřesům, popř. by nemohl spadnout. Jelikož detekce pohybu je implementována víceméně rozdílovou metodou, bylo třeba naimplementovat i funkci, kdy aplikace zjistí, zda dochází k otřesům zařízení. Pokud k těmto otřesům dochází, aplikace nemůže detekovat pohyb. Tato funkce je také implementována, aby nezachytila pohyb, při odebrání zařízení z místa, kde je postaven. Z tohoto nám vyplývá, že kamera může detekovat pohyb, pouze pokud je kamera v klidu.

Jelikož je obraz, kvůli snazší a přesnější detekci pohybu rozdělen do čtyř segmentů, tak se nám na levé straně objevil sloupec čísel, které značí následující: první až čtvrté číslo jsou procentuální shody prvního až čtvrtého segmentu a číslo páté nám udává, zda je mobil v pohybu, či nikoli a tudíž zda muže být pohyb detekován, nebo ne. Na obrázku 12 lze vidět režim střežení a jsou zde vyznačeny segmenty popsané číslem.



Obrázek 12: Režim střežení

Po uplynutí časového limitu, který je taktéž možný nastavit, začne aplikace střežit. Po započnutí střežení se nám na obrazovku vypíše také text „*Střežím...*“ a po detekci pohybu text „*Byl zaznamenán pohyb !*“. Následně se mohou vypisovat texty že se posílá E-mail,SMS atd. Ovšem jak procentuální shody, tak informační texty jsou pro uživatele většinou nepodstatné, jelikož předpokládáme, že u střežení tento uživatel není. Kvůli tomu, je i naimplementováno částečné zatmění obrazovky aplikace a také, aby aplikace nespotřebovávala velké množství energie z baterie zařízení.

Dále musíme mít na paměti důležitou věc. Předpokládejme, že zařízení (telefon) bude střežit zónu např. několik hodin. Naskýtá se zde možnost, že při střežení, přijde na telefon SMS zpráva, nebo se na telefon někdo bude snažit dovolat. Tím pádem začne mobil vyzvánět a je pak pro potencionálního zloděje snadnou kořistí. Toto bylo vyřešeno naprostým ztlumením zvuků. V případě volání, či nějakého jiného důvodu, kdy aktivita přejde do pozadí, aplikace usne a neprobíhá střežení. Ovšem po přejití aktivity opět do popředí (konec vyzvánění telefonu) začne aplikace opět střežit.

Po detekci pohybu, zařízení samozřejmě přestane snímat obraz z integrované kamery. Další implementovanou funkcí je to, že se do podsložky „*debug*“ uloží 2 snímky, mezi kterými byl zaznamenán pohyb. Dále se uloží soubor „*evaluation*“, což je textový soubor, ve kterém jsou zaznamenány podobnosti všech segmentů. Tato funkce je stálá a nelze ji v nastavení nijak měnit. Následně se vyfotí fotka střežené zóny a pak se dle nastavení provedou další události.

4.3.3 Alarm

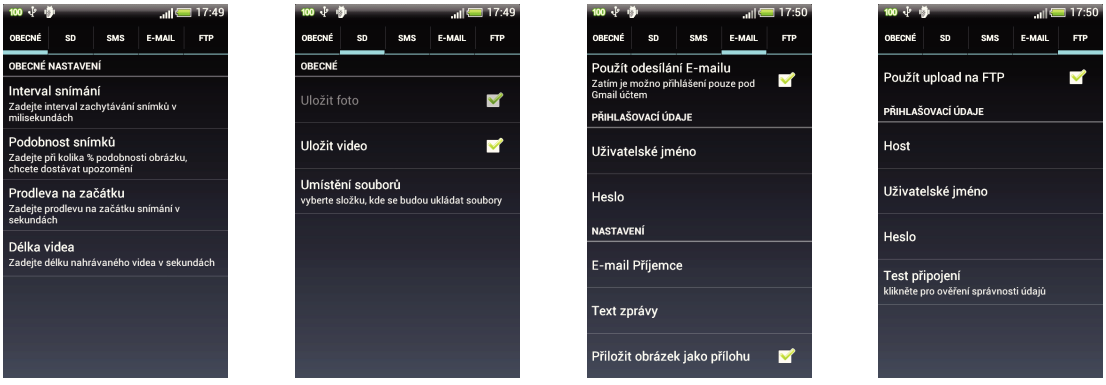
Spouští se tlačítkem Alarm a je to vlastně zjednodušená implementace střežení zóny. Pro spuštění střežení se stačí dotknout v kterémkoli místě obrazovky a aplikace opět začne odpočítávat čas a po odpočtu střežit. Zařízení (telefon) musí být opět postaveno na plochu, kde nedochází k žádným otřesům.

Nemáme zde už informační hodnoty, ani tlačítko spuštění. Také se neukládají fotky, mezi kterými byl detekován pohyb ani log, kde jsou uvedeny procentuální shody segmentů. Při detekci pohybu se pouze spustí siréna z reproduktoru zařízení.

U této funkce tedy předpokládáme, že vlastník mobilu, který si nechal zónu střežit, je poblíž a chce být informován pouze varovným signálem.

4.3.4 Nastavení

Menu nastavení je navrhnuto s vysokým důrazem na přehlednost. Grafické zpracování je zobrazeno na obrázku 13.



Obrázek 13: Nastavení aplikace

V horní části nastavení máme menu s následujícími položkami:

- Obecné

  V této záložce je možnost nastavení intervalu snímání, tj. časový interval mezi pořízením dvou porovnávaných snímků. Pak je zde také nastavení podobnosti snímků. To nám značí shodnost dvou porovnávaných snímků mezi sebou. Dále si zde můžeme nastavit prodlevu na začátku, což je úvodní odpočet, po kterém se spouští střežení a také délku nahrávaného videa v sekundách.

- SD

  Zde je možnost nastavení, zda chceme uložit foto (tato volba je prozatím nuceně povolena, jelikož kdyby nebyla povolena, nepošle se i automaticky E-mail s přílohou a neproběhne upload fotky na FTP server) a video. Je zde také možnost pojmenovat si složku, kde se budou tyto data ukládat.

- SMS

  Prvně je zde možnost, zda chceme vůbec dostávat SMS upozornění. Dále se vypisuje telefonní číslo příjemce a text zprávy SMS.

- E-mail

  Taktéž je zde možnost, zda chceme posílat E-mail. Pro správné odeslání E-mailu se musíme přihlásit do své E-mailové schránky (je potřeba vyplnit jméno a heslo v nastavení). Tato aplikace funguje prozatím jen pro přihlašování k Gmail účtu. Dále zde musíme vepsat E-mail příjemce a text zprávy. Předmět E-mailu už je automaticky předepsán na: „*Narušení střeženého objektu*“. Jako poslední možnost je zde přiložení vyfoceného obrázku jako přílohu k E-mailu.

- FTP

  Jako tomu bylo u předchozích dvou variant, prvně se nastaví, zda chceme použít upload na FTP. Dále se musí nastavit host, uživatelské jméno a heslo. Tato záložka je doplněna možností otestováním funkčnosti připojení k FTP serveru.

4.3.5 Informace

Při stisku posledního tlačítka se objeví jednoduchá obrazovka, kterou by si měl uživatel před použitím aplikace přečíst. Je zde stručně popsáno, jak aplikaci používat.

5 Implementace aplikace

Tato kapitola je věnována implementaci aplikace. Popíši zde vývoj aplikace, vytvořené aktivity a třídy a vysvětlím funkčnost vybraných algoritmů. [1, 2]

5.1 Úvodní obrazovka

Úvodní obrazovka se zobrazí při spuštění aplikace. Tato obrazovka, jakožto všechny obrazovky v systému Android, je implementována jako aktivita. To je třída, která vznikla zděděním od třídy `Activity`. Je to vlastně část aplikace, která je určena pro komunikaci a interakci s uživatelem. Jelikož jsou tyto aktivity odděleny od procesu aplikace, můžeme se mezi aktivitami jednoduše přesouvat. To je dosaženo pomoci intentů, což jsou vlastně zprávy, které si aktivity mezi sebou předávají.

Pak nám tato aktivita implementuje rozhraní `OnClickListener`, díky tomu můžeme zachytit dotek na obrazovce. Celá funkčnost této aktivity spočívá v tom, že jsou zde vytvořeny čtyři komponenty `ImageButton`. V případě stisku prvních třech se právě pomocí intentů přepneme do dalších aktivit. Při stisku posledního tlačítka (Info) je pouze zobrazen jednoduchý pohled s komponentou `TextView`, kde jsou vypsány stručné informace.

5.2 Režim střežení

Režim střežení je spuštěn při stisknutí tlačítka „*Spustit*“ na úvodní obrazovce. Je to také aktivita (`GuardActivity`), a jelikož má na starost víceméně celou funkčnost aplikace, bude rozdělena do několika následujících podkapitol.

5.2.1 Inicializace třídy

Aktivita `GuardActivity` se samozřejmě vytvoří pomocí intentu. Při vytvoření této aktivity se ihned zavolá metoda `onCreate(Bundle savedInstanceState)`. V této metodě si vytvoříme, popř. inicializujeme proměnné, které budeme ve třídě používat, vytvoříme nové instance, časovače atd. Zkrátka vše, co chceme po vytvoření této aktivity.

Jelikož aktivita střežení může být spuštěna až několik hodin, potřebujeme v co největší míře omezit odběr energie z baterie zařízení. To je řešeno nastavením minimálního jasu obrazovky a nastavením zámku, aby aplikace neusnula následovně:

```
PowerManager powerManager = (PowerManager) getSystemService(Context.POWER_SERVICE);
wakeLock = powerManager.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "Guard");
wakeLock.acquire();
```

Výpis 1: Nastavení režimu ztlumeného displeje

Dále je potřeba nastavit telefon do tichého režimu:

```
audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
audioManager.setRingerMode(AudioManager.RINGER_MODE_SILENT);
```

Výpis 2: Nastavení tichého režimu

Následně jsou v této metodě inicializovány všechny proměnné, do kterých se předá nastavení aplikace. Např. délka odpočtu, délka nahrávaného videa atd. Abychom na obrazovce mohli vidět reálný obraz, zachytávaný z integrované kamery, potřebujeme si prvně inicializovat komponentu `SurfaceView`, kde je tento obraz přenášen. Tato komponenta je zobrazena na celou obrazovku. Následně si musíme kameru otevřít (pokud jí zařízení disponuje) a poté vytvoříme novou a jedinou instanci třídy `CameraPreview`, která se nám stará o vykreslování obrazu v reálném čase.

Poté je zjištěn stav dostupnosti sítě. Pokud není k dispozici žádná Wi-Fi síť, pomocí třídy `Toast` vypíše aktivita na obrazovku, že neproběhne odeslání E-mailu nebo neproběhne upload na FTP server. Dále jsou zde vytvořeny 2 časovače:

- `recordTimer`

  Timer, který se spustí, při nahrávání videa. Při doběhnutí časovače se zavolá metoda `onFinish()`, která následně zavolá metodu `stopRecord()`, a ta zastaví nahrávání.

- `countDowntimer`

  Timer, který se spustí, při stisku tlačítka pro střežení. Každou minutu se volá metoda `onTick(long millisUntilFinished)`, která nám odečte vždy sekundu od odpočtu a vypíše ho na obrazovku. Po skončení odpočtu se vypíše do horní části obrazovky text: „*Střežím...*“ a spustí se Runnable rozhraní, které je popsáno níže.

Následně je vytvořeno tlačítko na střežení, které se po každém doteku mění na stav zap./vyp. a je zde implementován Listener na otřesy mobilu. Ten je realizován pomocí třídy `ShakeListener`. Metoda `onSensorChanged(int sensor,float[] values)` implementována v této třídě se volá při každé změně pohybu mobilu. Dále nám proměnná *sensor* udává senzor, který užíváme (v tomto případě akcelerometr) a proměnná *float[]* polohu akcelerometru v zařízení pomocí souřadnic x, y, z. Následně se přepočítavají aktuální a předchozí pozice a v případě překročení stanoveného práhu nám metoda zavolá metodu `onShake()`. Pohyb si následně v aktivitě zjistíme pomocí proměnné *shake*, která může nabývat hodnot *true/false*.

Nakonec je vytvořeno Runnable rozhraní, které se spouští v určitém intervalu. Tento interval v sobě nese proměnná *snapshotInterval* a lze ho nastavit v nastavení aplikace. Také je zde použita proměnná *canRun*, pomocí které se vypíná či zapíná běh tohoto rozhraní.

Pro nás je velmi podstatná metoda `cacheBitmaps`, která se v tomto rozhraní volá. Ta je popsána v další podkapitole.

```
Runnable runnable = new Runnable() {
    public void run() {
        if (canRun){
            cacheBitmaps();
            handler.postDelayed(runnable, snapshotInterval);
        }
    }
};
```

Výpis 3: Runnable rozhraní

5.2.2 Sejmutí snímků

Jak již bylo zmíněno, nejpodstatnější je pro nás metoda `cacheBitmaps()`. Ta se spouští v určitém intervalu. Porovnávají se zde vůči sobě dva snímky a můžou zde být vyvolány události jako nahrání videa, poslání SMS atd.

Při prvním průchodu metody se sejme aktuální snímek obrazovky. Je zavolána metoda `getBitmap()` ve třídě `CameraPreview`, která nám vrátí aktuální snímek obrazovky. Tato třída má v sobě vytvoření nového rozhraní `PreviewCallback`, která obsahuje metodu `onPreviewFrame(byte[] data, Camera camera)`. Ta se volá neustále, vždy při zobrazení předcházejícího snímku. Následně je v metodě proveden převod YUV dat, což je RAW formát Android zařízení, do barevného RGBA modelu, kde A značí průhlednost. Tento převod i s vytvořením bitmapy můžeme vidět na výpisu 4. Nejprve se dekódují YUV data, která jsou uložena v proměnné *data* a výsledné pixely v podobě celočíselných čísel, jsou vloženy do pole *pixels*. Z tohoto pole se následně vytvoří bitmapa s velikostí, která je dána výchozím nastavením zařízení (v mém případě 640x480), a o konfiguraci $ARGB\_8888$, která nastaví, že každý pixel je uložen do 4 bytů. Získáme tedy nejlepší možnou kvalitu bitmapy s osmi-bitovou přesností. Složka A je v tomto případě pevně nastavena na nejvyšší hodnotu, jelikož nedokážeme z kamery zachytit průhledný obraz. Nakonec je vytvořena nová zmenšená bitmapa na polovinu z původní bitmapy, jelikož při větším počtu pixelů, následně zařízení výpočetně nestíhá porovnávat všechny tyto pixely.

```
mCamera.setPreviewCallback(new PreviewCallback() {
    public void onPreviewFrame(byte[] data, Camera camera) {
        decodeYUV420SP(pixels, data, width, height);
        bitmap = Bitmap.createBitmap(pixels, width, height,  Bitmap.Config.ARGB_8888);
        bitmap = Bitmap.createScaledBitmap(bitmap, width/2, height/2, false);
    }
});
```

Výpis 4: Sejmutí obrazu

5.2.3 Porovnávání snímků

Jakmile je získán aktuální snímek obrazovky, zavolá se z aktivity `GuardActivity` metoda `init(bmp1)` ve třídě `CompareBitmaps`. Toto je provedeno pouze při prvním průchodu a nastaví se výška, šířka porovnávaných bitmap a deklaruje se 8 polí pro segmenty, pro každý snímek tedy 4 pole. Následně se z aktivity zavolá metoda `Compare(bmp1, bmp2)` ve třídě `CompareBitmaps`, ovšem jelikož se posílá pouze jeden snímek, algoritmus ho nemá s čím porovnat, tak se vrátí uměle vytvořená 100 % shoda. Ovšem při druhém průchodu je jako referenční snímek uložen snímek, který byl sejmut při průchodu prvním a je opět sejmut snímek aktuální. Nyní se můžou tyto snímky porovnat.

Při druhém průchodu máme k dispozici již dva snímky. Tyto 2 snímky (bitmapy) se pošlou jako parametry metodě `Compare(bmp1, bmp2)`. Pokud jsou tyto bitmapy nenulové, vloží se do polí `diff[x]`, kde x značí čísla segmentů od 0 do 3, hodnoty 0 a na konci srovnávání je zde přepočítán procentuální výsledek shody snímků.

Aby se mohly srovnávat bitmapy, vloží se barvy pixelů z bitmap do polí segmentů, které jsou označeny *sx_y*, kde *x* je číslo snímku a *y* číslo segmentu. Tato barva je uložena v poli jako celočíselné číslo a může nabývat hodnoty od -16777216 do -1. Jsou to vlastně všechny kombinace barev z RGB modelu. Ovšem tato hodnota nám nic neříká, proto si ji potřebujeme převést na RGB složky následovně:

```
private float [] IntToRGB(int pixel) {

    float red = Color.red(pixel);
    float green = Color.green(pixel);
    float blue = Color.blue(pixel);

    float [] RGB = {red, green, blue};
    return RGB;
}
```

Výpis 5: Převod z čísla na RGB složky

Porovnává se tedy každý pixel jedné bitmapy s korespondujícím pixelem bitmapy druhé. Tato metoda je nejvíc podobná metodě rozdílové, s tím rozdílem, že pixely se od sebe neodečítají, ale zkoumá se jejich „*podobnost.*“. Pod pojmem podobnost rozumíme vzdálenost dvou barevných bodů v Eukleidovském prostoru. Toto si můžeme představit, jako kdybychom 3 složky z RGB modelu dosadili do prostorových souřadnic x, y, z. Vznikne nám tedy RGB krychle. V této krychli pak počítáme vzdálenost dvou barevných bodů. Tento výpočet je uveden na výpisu 6, kde proměnné *RGB1[x]* a *RGB2[x]* jsou pole pro oba snímky, kde se ukládají RGB složky v podobě čísla 0-255;

```
private void CalculateDistance() {

    distance = Math.sqrt(((RGB1[0] − RGB2[0]) ∗ (RGB1[0] − RGB2[0]))
          + ((RGB1[1] − RGB2[1]) ∗ (RGB1[1] − RGB2[1]))
          + ((RGB1[2] − RGB2[2]) ∗ (RGB1[2] − RGB2[2])));
}
```

Výpis 6: Výpočet Eukleidovské vzdálenosti

Jakmile je vypočtena vzdálenost, porovnává se s předem určenou odchylkou vzdálenosti. Ta je nastavena v proměnné *SENS* na číslo 30 a byla určena na základě testování aplikace, které číslo je pro tuto vzdálenost vhodné. Rozumíme tedy, že pokud je v prostoru mezi body vzdálenost 30 a méně, je tato barva považována za shodnou.

Pokud je tedy barva pixelu první bitmapy shodná s barvou pixelu bitmapy druhé, přičte se číslo 1 do pole *diff[x]*, kde *x* značí číslo segmentu od 0 do 3. V poli máme následně počet shodných pixelů jednoho segmentu. To je následně přepočítáno na procenta a celé pole je vráceno třídě `GuardActivity`, kde se již podle předem nastaveného práhu určí, zda se jedná o pohyb, či nikoli.

Na výpisu 7 je zobrazeno kompletní porovnávání snímků. Kvůli obsáhlosti zdrojového kódu je uveden pouze výpis pro jeden segment každého snímku.

```
public int [] Compare(Bitmap b1, Bitmap b2){
    diff [0] = 100;

    if (b1 != null && b2 != null) {

        diff [0] = 0;

        b1.getPixels(s1_1, 0, width/2, 0, 0, width/2, height/2);
        b2.getPixels(s2_1, 0, width/2, 0, 0, width/2, height/2);

        for (int i = 0; i < area; i++) {

            RGB1 = IntToRGB(s1_1[i]);
            RGB2 = IntToRGB(s2_1[i]);
            CalculateDistance();
            if (distance < SENS)
            {
                diff [0]++;
            }

        }

        float ar = (area);
        diff [0] = Math.round((diff [0]/ ar)*100);

        return diff ;
    }
    return diff ;
}
```

Výpis 7: Porovnávání snímků

Na výpisu můžeme vidět metodu `Compare(Bitmap b1, Bitmap b2)`. Zprvu je nastavená 100 % shoda, která se vrátí, pouze v případě, kdy se předává pouze jedna nenulová bitmapa. Následně se do polí segmentů pomocí metody `getPixels()` vloží barvy pixelů. Nakonec se jen v cyklu zjistí, zda jsou si korespondující pixely podobné, a po ukončení cyklu se přepočítá pole *diff* na procentuální shodu, která je metodou vrácena.

Nakonec je ve třídě `GuardActivity` provedena kontrola, jestli je alespoň jedna procentuální shoda segmentu menší než předem nastavený práh. Pokud ano, začnou se provádět operace jako nahrávání videa, odesílání SMS atd. Toto porovnávání je uvedeno na následujícím výpisu:

```
if ((shake == false) && ((diff [0] < imagesSimilarity) || (diff [1] < imagesSimilarity) ||
        (diff [2] < imagesSimilarity) || (diff [3] < imagesSimilarity))) {
    ...
    saveBitmaps(bmp1, bmp2, diff);
    ...
}
```

Výpis 8: Porovnání shod u snímků

5.2.4 Události

Po vyhodnocení snímků se vyvolá metoda `saveBitmaps(bmp1, bmp2, diff)`. Ta nám uloží do složky *„debug“* 2 snímky, mezi kterými byl detekován pohyb a log, se shodností segmentů. Tyto snímky se ukládají vždy pouze 2 poslední. Po uložení se zavolá metoda `createTimeStamp()`, která vytvoří časové razítko. Toto časové razítko se použije jako název pro vyfocenou fotku. Také se nám vytvoří celá cesta k souboru. Ta se následně použije, pokud se bude odesílat foto na E-mail nebo FTP server.

```
private void createTimeStamp() {
  timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());

  if (checkMail || checkFTP) {
     File dir = new File (Environment.getExternalStorageDirectory().getPath() + "/" + sdPath);
      if (!dir.exists()){
          dir.mkdir();
      }
     attachmentFile = new File(dir.getPath() + "/IMG_"+ timeStamp + ".jpg");
  }
}
```

Výpis 9: Vytvoření časového razítka

Po vyfocení fotky se může nahrát video (pokud si to uživatel nastavil). To je nahráno pomocí třídy `MediaRecorder`. Kvalita videa je pevně stanovena na rozlišení 640x480 při 30 snímcích za sekundu.

Dále můžou nastat tyto události:

- Odeslání SMS

  SMS zpráva se odesílá pomocí třídy `SendSMS`. V konstruktoru třídy se pouze inicializují proměnné pro telefonní číslo a zprávu z nastavení. SMS se odešle při zavolání metody `sendSMS()`, je zde využito třídy `SmsManager`.

- Odesílání E-mailu

  Je řešeno pomocí dostupné knihovny *javamail-android* [1]. Je volně ke stažení pod licencí GNU GPL v2, což je licence pro svobodný software. Tato knihovna je napsaná v jazyce Java, je zde tedy velká výhoda přenositelnosti mezi různými platformami.

- Upload na FTP server

  Upload na FTP probíhá za použití knihovny *Apache Commons Net* [2], která implementuje funkčnost několika protokolů ze strany klienta. Např.: FTP, HTTP, NNTP, Telnet, NTP a další. Tato knihovna je také volně ke stažení pod licencí Apache, která je kompatibilní s licencí GPL.

Všechny tyto 3 události běží ve vlastním vlákně, aby nezatěžovaly hlavní vlákno aplikace. Je použito třídy `AsyncTask` a pokud jsou povoleny alespoň 2 události, jsou prováděny za sebou.

[1] Ke stažení na: `http://code.google.com/p/javamail-android/downloads/list`
[2] Ke stažení na: `http://commons.apache.org/net/download_net.cgi`

5.3 Režim alarmu

Režim alarmu je spuštěn při stisknutí tlačítka „*Alarm*“ na úvodní obrazovce. Je to jen zjednodušená aktivita, aktivity střežení, ovšem s několika rozdíly:

- Zvýšení hlasitosti

  Jelikož se dá předpokládat, že uživatel má nastaven tichý režim, nebo má zeslabenou hlasitost, je zde implementováno nastavení normálního režimu a poté zvýšení hlasitosti na maximum. Toho je dosaženo pomocí třídy `audioManager`.

- Spuštění střežení

  Spuštění střežení se již nespouští tlačítkem, ale dotykem v libovolném místě obrazovky. Toho je dosaženo pomocí metody `onTouchEvent(MotionEvent event)`.

- Přehrání sirény

  Je řešeno třídou `SoundPool`. Ta má tu výhodu, že se soubor *siren.mp3* načte do mezipaměti již při vytvoření aktivity `AlarmActivity` a při detekci pohybu se následně spouští pořád dokola.

Na výpise 10 vidíme metodu `initSound()` na inicializaci zvuku, kdy se načte mp3 soubor a metodu `playSound(int sound)`, přehrávající tento soubor, která je spuštěna při detekci pohybu.

```
private void initSound() {
    soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 100);
    soundPoolMap = new SparseIntArray();
    soundPoolMap.put(1, soundPool.load(this, R.raw.siren, 1));
    }

public void playSound(int sound) {
    AudioManager mgr = (AudioManager) this.getSystemService(Context.AUDIO_SERVICE);
    int streamVolume = mgr.getStreamVolume(AudioManager.STREAM_MUSIC);
    soundPool.play(soundPoolMap.get(sound), streamVolume, streamVolume, 1, -1, 1f);
}
```

Výpis 10: Přehrání sirény

5.4 Nastavení

Nastavení je realizováno aktivitou `PreferencesActivity`. Na horní záložky je využito třídy `TabHost`, kdy se po doteku na příslušnou sekci přepneme pomocí intentu do další příslušné aktivity. Jako výchozí je brána aktivita `MainSettings`. Následně jsou všechny aktivity v nastavení realizovány jako aktivity vzniklé zděděním od třídy `PreferenceActivity`. Jedná se tedy o třídy, využitelné přímo k nastavení aplikace. V každé této aktivitě je pouze pojmenování každého nastavení, případně jsou zde metody na zablokování možnosti nastavení, či testu pro připojení na FTP server.

6 Testování

Na konci vývoje mé aplikace byla tato testována z hlediska využitelnosti a výdrže. Také jsou v této kapitole uvedeny možné problémy a jejich řešení.

Ke všem testům jsem měl k dispozici 3 telefony. Můj vlastní HTC Desire S s verzí 4.0.4 Ice Cream Sandwich, pro který byla aplikace primárně vyvíjena. Dále Samsung Galaxy Ace 2 ve verzi 2.3.6 Gingerbread a Samsung Galaxy Mini 2 ve verzi 2.3.6 Gingerbread.

6.1 Výdrž a spotřeba

Jako jeden z aspektů byla aplikace podrobena testu výdrže baterie. Tento test byl zkoušen postavením telefonů na klidné místo, kde nedocházelo k žádným otřesům a kde nemohl být detekován žádný pohyb. Následně bylo spuštěno střežení zóny. Ke zjištění aktuální spotřeby byl použit program *Battery Monitor Widget*. Ze všech hodnot, která aplikace zaznamenávala každou minutu, byla nakonec vypočtena průměrná spotřeba aplikace.

V tabulce 2 je možno vidět testované telefony, jejich kapacitu baterie, následně spotřebu, odhadovanou a reálnou výdrž telefonu.

	HTC Desire S	Samsung Galaxy Ace 2	Samsung Galaxy Mini 2
Kapacita baterie	1450 mAh	1500 mAh	1300 mAh
Průměrná spotřeba	178 mA	247 mA	225 mA
Odhadovaná výdrž	8:08 h	6:00 h	5:42 h
Reálná výdrž	6:20 h	5:30 h	5:35 h

Tabulka 2: Výdrž a spotřeba

6.2 Využitelnost a rychlost

Dále byla aplikace zhodnocena z hlediska její využitelnosti a rychlosti.

Aplikace byla vyvíjena pouze jako ukázka, jak lze vytvořit jednoduchou bezpečnostní kameru na telefonu, či jiném přenosném zařízení, postavené na platformě Android. Jak již bylo uvedeno v úvodu, existuje již několik aplikací postavených na obdobném principu, mám již tedy nějakou konkurenci. Tato aplikace je spíše vhodná pro domácí využití, pokud si někdo chce pohlídat soukromé věci apod.

Při testování aplikace jsem nadále zjišťoval její rychlost a plynulost. Na všech třech telefonech běžela aplikace plynule a při výchozím nastavení jsem nenašel žádné problémy s rychlostí. Aplikace je tedy co do použitelnosti plynulá.

Také jsem zjišťoval rychlost posílání E-mailu a rychlost posílání dat na FTP server. Aplikace umožňuje prozatím odesílání obrázku v sítí Wi-Fi, jelikož tyto sítě jsou dnes již skoro všude a předpokládám, že místo ke střežení bude pokryto Wi-Fi síti. K dispozici jsem měl Wi-Fi síť o konektivitě 14 Mbps. E-mail byl odeslán během několika sekund, ovšem upload na FTP trval okolo minuty. Bylo zjištěno, že toto je neefektivní implementací knihovny *Apache Commons Net*.

6.3 Problémy a jejich řešení

Při testování aplikace jsem nenarazil na žádné chyby. Tím ovšem není zaručeno, že tato aplikace bude správně fungovat na všech zařízeních, jelikož jsem ji testoval pouze na třech mobilních telefonech. V aplikaci jsou ovšem 2 nevýhody:

- Ukončení střežení

  Při stisknutí tlačítka Power se ukončí střežení a při opětovném stisknutí tlačítka se musí střežení spustit znova. Touto chybou jsem se dále nezaobíral, jelikož předpokládám, že uživatel není u zařízení, a tudíž nemůže toto tlačítko zmáčknout.

- Pauznutí střežení

  Dalším možným problémem je, že pokud aplikace střeží a někdo se na telefon bude snažit dovolat, tak se aplikace přesune do pozadí a nebude střežit zónu. Po dokončení volání začne aplikace opět střežit.

Také jsem zkoušel zachytit několik falešných pohybů. Tento falešný pohyb může nastat ze dvou důvodů:

- Pohyb zařízení

  Tento falešný pohyb je možno zachytit, pokud se zařízení (telefon) pohne a kvůli nastavení citlivosti ho akcelerometr nezaznamená. Kdyby se telefon pohnul, kamera by sejmula druhý snímek odlišný od prvního snímku. V závěru by se tedy nesrovnávaly stejné snímky, tudíž korespondující pixely v obraze. Ovšem pokud uživatel postaví telefon na místo, kde nedochází k otřesům, aplikace žádný falešný pohyb nezaznamená.

- Špatné nastavení

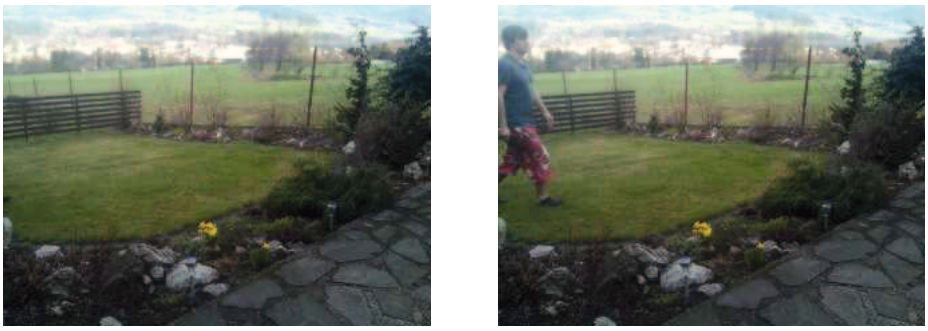
  Toto je dáno špatným nastavením procentuální podobností snímků. Pokud by si uživatel tento práh nastavil např. na 98%, je zde možnost, že by mohl být detekován falešný pohyb. Ten by byl pravděpodobně detekován kvůli šumu, který by byl v obraze.

Dále se zde naskýtá možnost, že je pohyb vykonán, ale není aplikací zaznamenán. To může být neúměrným nastavením intervalu snímání, nebo podobnosti snímků. Například pokud zvolíme malý interval snímání, teoreticky by mohla osoba před kamerou projít, jelikož na každém snímků by byla pouze o kousek dále než na předchozím, a tudíž by byla procentuální shoda snímků velmi malá. Naopak pokud bychom zvolili velký interval snímání, osoba by nemusela být vůbec zachycena, kdyby se v obraze přemístila přesně mezi sejmutím dvou snímků. Po testování jsem došel jako k ideálnímu nastavení této hodnoty na hodnotu 300 ms. Dalším možným nastavením je podobnost snímků. Tato podobnost se může v různě osvětlených místnostech různě měnit, tudíž je doporučeno si aplikaci odzkoušet v režimu alarmu. Ovšem po testování jsem došel k doporučené hodnotě 75%.

6.4 Ukázky detekce pohybu

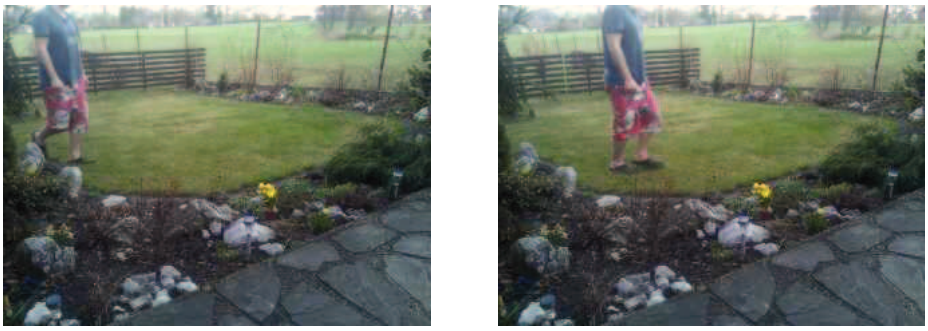
Nakonec bylo provedeno několik správných detekcí.

Na obrázku 14 vidíme referenční a aktuální snímek obrazovky. Srovnáním těchto dvou snímků aplikace detekovala pohyb. V aplikaci bylo nastaveno 90% podobnosti snímků a procentuální shody segmentů jsou: 1 - 87%, 2 - 100%, 3 - 97%, 4 - 100%.



Obrázek 14: Správná detekce pohybu 1

Na obrázku 15 můžeme vidět taktéž správně zaznamenanou detekci pohybu. Nastaveno bylo již 75% podobnosti snímků. K dispozici jsou také procentuální shody segmentů: 1 - 68%, 2 - 100%, 3 - 99%, 4 - 100%.



Obrázek 15: Správná detekce pohybu 2

Z obrázku můžeme usoudit, že při 90% podobnosti snímků, byl detekován pohyb téměř okamžitě. Při 75% byla očividně detekce zachycena až později. To je dáno tím, že na přechozím snímku neklesla procentuální podobnost pod 75%.

7 Závěr

Cílem této bakalářské práce bylo zjistit některé možné algoritmické řešení detekce pohybu. Následně jsem měl za úkol vyhledat již existující aplikace pro platformy Windows Phone, iOS a Android, pracující na některém z těchto principů. Ovšem hlavním úkolem bylo vymyslet a naimplementovat vlastní aplikaci, sloužící jako jednoduchá bezpečností kamera.

V úvodu uvádím některé dostupné aplikace, testuji je a nakonec porovnávám s mojí vytvořenou aplikací. Následně podávám vysvětlení principu zaznamenávání obrazu do digitální podoby, uvádím jak se tento obraz reprezentuje a stručně shrnuji některé barevné modely. Poté vysvětluji, jak je možné vůbec zachytit pohyb v obraze a uvádím některé jednoduché metody na detekci pohybu v obraze. Nakonec popisuji mojí aplikaci z hlediska uživatele. Uvádím, jak aplikaci požívat, jaké má funkce a možnosti nastavení. Na to navazuji kapitolou, jak jsem vůbec aplikaci vytvářel a vypsal jsem z mého hlediska důležité výpisy zdrojového kódu. Nakonec jsem aplikaci testoval, zhodnocoval výdrž a uvedl možné problémy aplikace, a jejich možné řešení.

Tato bakalářská práce mi byla velikým přínosem. Jednak jsem se více naučil programovat v Javě pro platformu Android, a také jsem se něco dozvěděl o reprezentaci digitálního obrazu a možného určení pohybu v něm.

Aplikaci bych chtěl následně distribuovat ve službě Google Play a podle ohlasů také přidávat různá nová vylepšení a funkce. Mezi taková vylepšení by mohly patřit funkce jako: posílání MMS, upload snímků na různé internetové služby a portály, streamování videa přes HTTP protokol a další. Jelikož aplikace hodnotí procentuální shodu snímků, bylo by možné přidávat různé algoritmy, porovnávající obraz postavené na podobném principu. Následně by si mohl uživatel zvolit metodu porovnávání např. dle rychlosti zpracovávání.

8 Reference

[1] Android SDK | Android Developers. [online]. 2012 [cit. 2013-01-12]. Dostupné z: http://developer.android.com/develop/index.html.

[2] MURPHY, Mark L. *Android 2: průvodce programováním mobilních aplikací*. Vyd. 1. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.

[3] LEE, H. – CHUVYROV, E. *Beginning Windows Phone App Development*. Apress, 2012. 548 s. ISBN 978-14-302-4134-8. Kapitola 1, Introducting Windows Phone and the Windows Phone Platform, s. 1–13.

[4] MARK, D. – NUTTING, J. – LAMARCHE, J. – OLSSON, F. *Beginning iOS 6 Development: Exploring the iOS SDK*. Apress, 2012. 800 s. ISBN 978-14-302-4512-4. Kapitola 1, Welcome to he Jungle, s. 1–10.

[5] UJBÁNYAI, M. *Programujeme pro Android*. Grada Publishing a.s., 2012. 187 s. ISBN 978-80-247-3995-3. Kapitola 1, Android - představení, s. 13-22.

[6] PETŘÍK, Oldřich. *Barevná korekce digitálního obrazu* [online]. Plzeň, 2007 [cit. 2013-04-19]. Dostupné z: http://graphics.zcu.cz/research/students/BP_2007_Petrik_Oldrich.pdf. Bakalářská práce. Západočeská univerzita v Plzni.

[7] BAJGAR, A. *Segmentačních techniky zpracování obrazu* [online]. Brno, 2011 [cit. 2013-04-19]. Dostupné z: http://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=39096. Diplomová práce. Vysoké učení technické v Brně.

[8] ŠEVČÍK, P. *Analýza podpisového vzoru s využitím umělé neuronové sítě* [online]. Brno, 2009 [cit. 2013-04-19]. Dostupné z: http://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=18116. Bakalářská práce. Vysoké učení technické v Brně.

[9] ZÍTKA, M. *Detekce pohybu v obraze* [online]. Brno, 2008 [cit. 2013-04-19]. Dostupné z: http://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=8263. Bakalářská práce. Vysoké učení technické v Brně.

[10] Detekce pohybu ve videu a jejich identifikace [online]. [cit. 2013-04-19]. Dostupné z: http://www.fbmi.cvut.cz/files/predmety/3528/public/Detekce%20pohybu%20ve%20videu.pdf.

A Přílohy na CD-ROM

Příloha 1

Text této bakalářské práce v elektronické podobě.

Příloha 2

Zdrojové kódy aplikace.

Příloha 3

Spustitelná aplikace pro Android.